1. (**CURRENTLY AMENDED**) A memory for storing a computer-implemented shared locking data store for handling multiple executable entities' access to at least one resource, said shared locking data store being stored in a computer memory, said shared locking data store comprising:

write requested data that is indicative of when a write request for the resource has been made;

writer active data that is indicative of whether a writer process is actively undergoing a write operation with respect to the resource;

wait event posted data, wherein the wait event posted data is indicative of whether a read request to the resource has completed, wherein the wait event posted data is used to indicate when a write request can access the resource;

reader data that is indicative of whether a reader process is active and attempting to read from the resource;

wherein the shared locking data store allows writer and reader locking state information to be determined so that access to the resource can be handled;

wherein encapsulation of both lock status data and the reader data in the shared locking data store allows a hardware atomic operation to operate upon both the lock status data and the reader data as a single unit for determining how access to the resource is to be handled;

wherein the multiple executable entities' access is a concurrent access to the at least one resource;

wherein a set status for the write requested data indicates whether an operating system (OS) mutex lock is to be used for processing access to the resource and is used to avoid an OS mutex lock when no writer process is pending.

2. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 1, wherein the executable entities include processes, threads, daemons, applets, servlets, ActiveX components, stand-alone applications, code that runs at the request of another program, or combinations thereof;

wherein use of the shared locking data stores allows for the avoidance of an operating system call or avoidance of a resource accessing trap of the operating system's kernel.

3. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 1, wherein the multiple executable entities include threads which are to access the resource, wherein the threads comprise a writer thread and a reader thread, wherein the locking state information is used to determine how the writer thread and the reader thread access the resource.

4. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 3, wherein the resource requires protection on a per thread basis.

5. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 3, wherein the locking state information indicates where in the overall locking process an operating system is with respect to the resource.

6. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 3, wherein the request of the reader thread is allowed to succeed unless there is a write request active or pending as indicated by the write requested data and the writer active data.

7. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 3 further comprising:

rules mechanism that indicates how resource access requests are to be handled based upon the data stored in the shared locking data store.

8. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 7, wherein the rules mechanism includes allowing any number of read requests to succeed unless there is a writer active or pending as indicated by the shared locking data store.

9. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 8, wherein the rules mechanism includes allowing only one writer to be active at any given time.

10. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 9, wherein the rules mechanism includes that no preference is given when deciding which of waiting read or write requests should be honored first.

11. **(PREVIOUSLY PRESENTED)** The memory for storing the shared locking data store of claim 10, wherein the rules mechanism substantially eliminates a reader request starvation or a writer request starvation situation with respect to the resource.

12. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein the resource comprises data stored in computer memory.

13. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein the resource comprises a computer file.

14. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein the resource comprises an input/output device.

15. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein the write requested data is to have a set status when a write request has been made.

16. (**CANCELED**)

17. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein indication of whether a writer process is active by the writer active data is used to protect against readers posting an operating system (OS) event after the writer thread has been activated.

18. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1 further comprising wait event posted data, wherein the wait event posted data is indicative of whether a read request to the resource has completed, wherein the wait event posted

data is used to indicate when a write request can access the resource, wherein a last active read clears status of the wait event posted data prior to posting an event.

19. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 18, wherein the clearing ensures that one and only one posting to a writer of an event occurs.

20. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 19, wherein the posting occurs if status of the writer active data is not set.

21. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 18, wherein the processes comprise threads, wherein the wait event posted data indicates when events can be posted by one thread to notify another thread that the other thread's request can be processed.

22. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 21, wherein the wait event posted data indicates whether a reader thread can post a lock release event to a writer thread.

23. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 22, wherein the wait event posted data is used to prevent multiple reader threads from posting redundant lock release events.

24. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 18, wherein the write requested data, writer active data, and wait event posted data provide multi-threaded protection in place of an operating system (OS) mutex.

25. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 18, wherein locking state data comprises the write requested data, the writer active data, the reader count data, and the wait event posted data;

wherein the locking state data is formatted as a single atomic unit.

26. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 25, wherein the locking state data is formatted as a multi-bit single unit.

27. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 26, wherein the formatting of the locking state data as a single unit allows an operating system to use atomic operations upon the locking state data.

28. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 27, wherein the atomic operations include an atomic get operation.

29. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 27, wherein the atomic operations include an atomic set operation.

30. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 27, wherein the atomic operations include an atomic add operation.

31. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 27, wherein the atomic operations include an atomic sub operation.

32. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 25, wherein the locking state data is formatted as a multi-bit single unit;

wherein the write requested data comprises a single bit in the unit;

wherein the writer active data comprises a single bit in the unit;

wherein the wait event posted data comprises a single bit in the unit;

wherein the reader data comprises a plurality of bits in the unit to indicate number of active readers with respect to the resource.

33. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 32, wherein the reader data comprises low order bits of the unit.

34. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 33, wherein the reader data's count of the readers provides an indication as to whether readers are present and when the last reader exits.

35. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 32, wherein a read lock acquisition means accesses the locking state data for processing a read lock acquisition of the resource.

36. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 32, wherein a read lock release means accesses the locking state data for processing a read lock release of the resource.

37. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 32, wherein a write lock acquisition means accesses the locking state data for processing a write lock acquisition of the resource.

38. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 32, wherein a write lock release means accesses the locking state data for processing a write lock release of the resource.

39. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein the operating system returns a lock busy status indicator when a lock cannot be obtained within a predetermined time.

40. (**CURRENTLY AMENDED**) A computer-implemented method for handling multiple executable entities' access to at least one resource, comprising:

receiving a request from an executable entity to access a resource;

determining how to process the resource request based upon lock state data;

wherein lock state data comprises write requested data and reader data;

wherein the write requested data is indicative of when a write request for the resource has been made with respect to the resource;

wherein if the write requested data indicates that a write request has not been made for the resource, then providing access to the resource such that an operating system (OS) mutex is avoided for a read lock acquisition;

wherein the reader data is indicative of whether a reader executable entity is active and attempting to read from the resource;

wherein encapsulation of both lock status data and the reader data in the shared locking data store allows a hardware atomic operation to operate upon both the lock status data and the reader data as a single unit for determining how access to the at least one resource is to be handled;

wherein the multiple executable entities' access is a concurrent access to the at least one resource;

wherein a set status for the write requested data indicates whether an operating system (OS) mutex lock is to be used for processing access to the resource and is used to avoid an OS mutex lock when no writer process is pending.

41. (**CURRENTLY AMENDED**) A computer-implemented apparatus for handling multiple executable entities' access to at least one resource, comprising:

means for receiving a request from an executable entity to access a resource;

means for determining how to process the resource request based upon lock state data;

wherein lock state data comprises write requested data and reader data,

wherein the write requested data is indicative of when a write request for the resource has been made with respect to the resource;

wherein if the write requested data indicates that a write request has not been made, then providing access to the resource such that an operating system (OS) mutex is avoided for a read lock acquisition;

wherein the reader data is indicative of whether a reader executable entity is active and attempting to read from the resource;

wherein encapsulation of both lock status data and the reader data in the shared locking data store allows a hardware atomic operation to operate upon both the lock status data and the reader data as a single unit for determining how access to the at least one resource is to be handled;

wherein the multiple executable entities' access is a concurrent access to the at least one resource;

wherein a set status for the write requested data indicates whether an operating system (OS) mutex lock is to be used for processing access to the resource and is used to avoid an OS mutex lock when no writer process is pending.

42. (**ORIGINAL**) The apparatus of claim 41 further comprising:

read lock acquisition means for accessing the lock state data in order to process a read lock acquisition of the resource;

read lock release means for accessing the lock state data in order to process a read lock release of the resource;

write lock acquisition means for accessing the lock state data in order to process a write lock acquisition of the resource;

write lock release means for accessing the lock state data in order to process a write lock release of the resource.

43. (**CURRENTLY AMENDED**) A machine-readable medium or media having software instructions for handling multiple executable entities' access to at least one resource, said software instructions, when executed by a machine, cause the machine to:

receiving a request from an executable entity to access a resource;

determining how to process the resource request based upon lock state data;

wherein lock state data comprises write requested data and reader data;

wherein the write requested data is indicative of when a write request for the resource has been made with respect to the resource;

wherein if the write requested data indicates that a write request has not been made for the resource, then providing access to the resource such that an operating system (OS) mutex is avoided for a read lock acquisition;

wherein the reader data is indicative of whether a reader executable entity is active and attempting to read from the resource;

wherein encapsulation of both lock status data and the reader data in the shared locking data store allows a hardware atomic operation to operate upon both the lock status data and the reader data as a single unit for determining how access to the at least one resource is to be handled;

wherein the multiple executable entities' access is a concurrent access to the at least one resource;

wherein a set status for the write requested data indicates whether an operating system (OS) mutex lock is to be used for processing access to the resource and is used to avoid an OS mutex lock when no writer process is pending.

44. (**PREVIOUSLY PRESENTED**) The memory for storing the shared locking data store of claim 1, wherein the hardware atomic operation includes machine level instructions for performing an operation on the shared locking data store for handling access to the resource;

wherein the reader data includes reader count data;

wherein the encapsulation of both the lock status data and the reader count data in the shared locking data store include encapsulation of status bits and bits which form an integer subset containing the count of reader requests that have been granted and that need to be processed;

wherein the status bits include a write pending-requested indicator that is set when a write request has been made in order to determine whether an OS mutex lock is required.